```
package esprreali;

// Questa classe eredita tutti i metodi non astratti di
// EsrpReale. Quindi, la classe Costante ha automaticamente
// i metodi stessaClasse e clone (che fa una copia
// superficiale). In particolare, il metodo stessaClasse va
// invocato come this.stessaClasse(o), ossia usando this e
// non super.

// Per i nomi delle classi, ci atteniamo alla convenzione
// dei nomi maiuscoli e al singolare: gli oggetti "Costante"
// rappresentano le costanti.

public class Costante extends EsprReale {
  private double valore;

        // anche se non e' specifiato esplicitamente nel testo del
        // problema, e' chiaro che occorre poter scrivere e
        // leggere il valore della costante; la soluzione piu'
        // pulita e' quella di scrivere usando il costruttore e
        // leggere con un metodo get
  public Costante(int v) {
    valore=v;
  }

  public double getValore() {
    return this.valore;
  }

        // il testo del problema specifica che la versione stringa
        // di una costante e' la stringa in cui c'e' solo il
        // valore della costante; scrivere quindi un metodo
        // con un corpo: return "Costante: "+valore; e' un errore
  public String toString() {
    return ""+valore;
  }

        // possiamo usare il metodo ausiliario di confronto
        // delle classi; si poteva anche fare il confronto
        // esplicitamente in questo metodo; in generale, nel
        // metodo equals non va fatto il confronto usando
        // instanceof, perche' (o instanceof Costante)
        // darebbe false in tutte le sottoclassi di
        // Costante, per cui poi il metodo equals andrebbe
        // ridefinito nelle sottoclassi anche se non sono
        // state aggiunte componenti all'oggetto; in
        // generale, a meno che una classe non sia
        // dichiarata final (cosa che non e' prevista dal
        // testo del problema) si deve sempre pensare che si
        // possano dover fare delle sottoclassi
  public boolean equals(Object o) {
    if(!this.stessaClasse(o))
      return false;

    Costante c=(Costante) o;

    return this.valore==c.valore;
  }

        // l'implementazione di hashCode che consiste nella sola
        // istruzione "return 7;" e' di scarso valore; lo stesso
        // vale per le implementazioni che troncano il valore
        // reale a intero (2.1 e 2.8 avrebbero lo stesso
        // hashCode); meglio quindi affidarsi al metodo
        // hashCode previsto per i reali, che si trova nella
        // classe Double
  public int hashCode() {
    Double b=new Double(this.valore);
    return b.hashCode();
  }
```

```
        // qui la clonazione superficiale va bene perche'
        // l'oggetto Costante non ha componenti oggetto;
        // possiamo quindi non ridefinire clone() e usare
        // direttamente quello ereditato da EsprReale
}
```

```
package esprreali;

// Questa classe rappresenta tutti gli oggetti di tipo
// "espressione di reali". Dal momento che non e' possibile
// realizzare toString, equals e hashCode a questo livello,
// non vengono ridefiniti. La classe contiene quindi solo
// metodi ausiliari (concreti) e il metodo clone (vedi
// sotto).

// NOTA IMPORTANTE: nella sovraclassi (come questa) non deve
// apparire in nessun modo il nome delle sottoclassi; per
// esempio, scrivere il metodo toString qui, facendo tutti i
// vari casi (if(o instanceof Costante)...) e' un errore
// grave; se si volesse per esempio aggiungere una nuova
// sottoclasse (per esempio, Differenza), bisognerebbe ogni
// volta andare a modificare la sovraclasse.

// L'interfaccia Cloneable viene implementata nella
// sovraclasse: in questo modo, tutte le sottoclassi
// implementano automaticamente questa interfaccia. Il
// metodo clone va necessariamente ridefinito in questa
// classe perche', per definizione, si deve poter fare
// e.clone() quando "e" e' una variabile di tipo EsprReale.
// L'eccezione CloneNotSupportedException va catturata nel
// clone di questa classe per lo stesso motivo.

public abstract class EsprReale implements Cloneable {
        // metodo ausiliario di confronto classi, da usare
        // nelle sottoclassi all'interno del loro metodo
        // equals; la protezione e' a livello di package,
        // dal momento che si tratta di un metodo che deve
        // essere usato solo all'interno delle altre classi
        // del package; e' un errore chiamare "equals" un
        // metodo fatto in questo modo, perche' equivale a
        // dire che, di default, due oggetti EsprReale sono
        // uguali se sono della stessa classe
  boolean stessaClasse(Object o) {
    if(o==null)
      return false;

    return this.getClass()==o.getClass();
  }

        // clonazione superficiale; in questo classe, la
        // clonazione non si puo' definire profonda, perche'
        // altrimenti bisognerebbe fare riferimento alle
        // sottoclassi; d'altra parte, il metodo clone deve
        // sempre poter essere invocato su un oggetto che
        // sta in una variabile EsprReale, e quindi il
        // metodo deve necessariamente ridefinita public a
        // questo livello, e l'eccezione va catturata
        // necessariamente a questo livello
  public Object clone() {
    try {
      return super.clone();
    }
    catch(CloneNotSupportedException e) {
      return null;
    }
  }
}
```

```
package esprreali;

// dato che usiamo le liste...
import java.util.*;

public class Prodotto extends EsprReale {
  private LinkedList m;

        // il testo non specifica come va creato un prodotto;
        // si capisce, d'altra parte, che deve essere in qualche
        // modo possibile creare un prodotto fatto di una sequenza
        // qualsiasi di espressioni
  public Prodotto() {
    m=new LinkedList();
  }
        // dal momento che il parametro e' EsprReale, siamo
        // sicuri che il tipo degli oggetti della lista saranno
        // tutti di tipo EsprReale; d'altra parte, va controllato
        // che non vengano inseriti valori null; la presenza di
        // null in una lista non e' un errore a priori; lo e'
        // perche' il testo esclude (anche se non esplicitamente)
        // che una espressione possa valere null
  public void addEspressione(EsprReale e) {
    if(e==null)
      throw new RuntimeException("Le espressioni non possono valere null");
    m.add(e);
  }

        // deve esistere un modo per determinare gli elementi che
        // fanno parte di un prodotto; rendere public la lista m
        // e' una soluzione non ideale; meglio definire un metodo
        // che trova il singolo elemento della lista
  public EsprReale get(int i) {
    return (EsprReale) m.get(i);
  }

        // serve anche un metodo per determinare la lunghezza
        // della lista; una alternativa migliore a questo metodo e
        // al precedente e' la definizione di un iteratore per il
        // prodotto
  public int size() {
    return m.size();
  }

        // il metodo toString deve restituire la concatenazione
        // delle espressioni, con * in mezzo; c'e' una piccola
        // complicazione: il fatto che + non va messo dopo
        // l'ultima espressione; usare un ciclo for con
        // m.get() andava bene, ma era meno efficiente
  public String toString() {
    String r="";
    Iterator i;

    i=m.iterator();

    while(i.hasNext()) {
      r=r+i.next().toString();
      if(i.hasNext())
        r=r+"*";
    }

    return r;
  }

        // il metodo equals di LinkedList fa il confronto degli
        // oggetti delle due liste, usando equals per fare questo
        // confronto; si puo' quindi definire equals di Prodotto
        // semplicemente invocando equals delle LinkedList; non e'
        // sbagliato fare un ciclo e confrontare gli elementi
  public boolean equals(Object o) {
```

```java
    if(!this.stessaClasse(o))
      return false;

    Prodotto p=(Prodotto) o;

    return this.m.equals(p.m);
  }

        // il metodo hashCode si ottiene come somma pesata; vedere
        // anche il commento nella classe Somma; non serve
        // fare il cast di i.next(), dato che hashCode e'
        // definito pubblico in Object
  public int hashCode() {
    int ris=5;

    Iterator i=m.iterator();

    while(i.hasNext())
      ris=ris*7+i.next().hashCode();

    return ris;
  }

        // la clonazione profonda di una lista prevede la
        // creazione di una nuova lista in cui vanno messe
        // le copie di tutti gli oggetti della lista di
        // partenza; usiamo comunque super.clone() per i
        // motivi spiegati nella classe Somma; notare che il
        // cast di i.next() e' necessario, dato che clone e'
        // protetto in Object
  public Object clone() {
    Prodotto p=(Prodotto) super.clone();
    p.m=new LinkedList();

    Iterator i=m.iterator();
    while(i.hasNext()) {
      EsprReale e=(EsprReale) i.next();
      p.m.add(e.clone());
    }

    return p;
  }
}
```

```
package esprreali;

public class Somma extends EsprReale {
  private EsprReale e1;
  private EsprReale e2;

        // il costruttore deve necessariamente avere due
        // argomenti, dato che viene specificato che la somma ha
        // sempre due addendi (mai zero oppure uno). Il
        // testo del problema specifica che le due
        // componenti di una somma sono due espressioni,
        // ossia non possono valere null; poter controllare
        // che non vengano inseriti valori sbagliati nelle
        // componenti di un oggetto e' uno dei vantaggi
        // dell'incapsulamento
  public Somma(EsprReale a, EsprReale b) {
    if((e1==null)||(e2==null))
      throw new RuntimeException("Le espressioni non possono valere null");
    e1=a;
    e2=b;
  }

        // deve essere possibile accedere alle due componenti di
        // una somma
  public EsprReale prima() {
    return e1;
  }
  public EsprReale seconda() {
    return e2;
  }

  // il metodo toString deve restituire esattamente quello
  // che viene richiesto: le due espressioni con + in mezzo
  public String toString() {
    return e1.toString()+"+"+e2.toString();
  }

        // il confronto va fatto nel modo standard: dato che le
        // due componenti di questa classe sono oggetti, vanno
        // confrontati con equals; qui sotto viene riportato il
        // metodo equals generale, ma in questo caso le due
        // componenti non possono valere null per come e'
        // fatto il costruttore;
  public boolean equals(Object o) {
    if(!this.stessaClasse(o))
      return false;

    Somma s=(Somma) o;

    if(this.e1==null) {
      if(s.e1!=null)
        return false;
    }
    else if(!this.e1.equals(s.e1))
      return false;

    if(this.e2==null) {
      if(s.e2!=null)
        return false;
    }
    else if(!this.e2.equals(s.e2))
      return false;

    return true;
  }

        // questo e' il modo corretto di implementare hashCode;
        // fare la somma non pesata degli hashCode delle
        // componenti e' un errore (piccolo, ma e' un errore).
        // Fare attenzione: se invece di ris=ris*7+ si fa
```

```
        // ris=ris+7*..., allora si sta facendo la somma (anche se
        // poi viene moltiplicata per 7), e non la somma pesata
  public int hashCode() {
    int ris=5;

    ris=ris*7+e1.hashCode();
    ris=ris*7+e2.hashCode();

    return ris;
  }

        // per la clonazione di Somma ci atteniamo al metodo
        // generale di clonare usando il metodo della
        // sovraclasse e poi cambiare le componenti; in
        // questo caso, puo' sembrare non necessario, dato
        // che poi tutte e due le componenti vanno cambiate;
        // invece va fatto, perche' altrimenti un
        // cambiamento nella sovraclasse (per esempio,
        // l'aggiunta di una componente) renderebbe non
        // funzionante questo metodo; in generale, le classi
        // dovrebbere essere il piu' possibile indipendenti
        // fra loro
  public Object clone() {
    Somma s=(Somma) super.clone();
    s.e1=(EsprReale) e1.clone();
    s.e2=(EsprReale) e2.clone();
    return s;
  }
}
```

```java
package esprreali;

public class Variabile extends EsprReale {
  private String nome;

        // si deve poter accedere al nome della variabile
  public Variabile(String n) {
    nome=n;
  }

  public String getNome() {
    return nome;
  }

        // va restituito il nome e basta, non una concatenazione
        // del tipo "Variable: "+nome. Vedi il commento in
        // Costante.java
  public String toString() {
    return nome;
  }

        // vedi anche il metodo equals di Costante.java; dal
        // momento che la componente nome e' di tipo oggetto, il
        // confronto va fatto usando il metodo equals (di String)
        // e non ==
  public boolean equals(Object o) {
    if(!this.stessaClasse(o))
      return false;

    Variabile v=(Variabile) o;

    return nome.equals(v.nome);
  }

  public int hashCode() {
    return nome.hashCode();
  }

        // dal momento che le stringhe non sono clonabili, ha
        // senso fare una clonazione senza replicare il nome delle
        // variabile; in questo caso, si puo' non ridefinire
        // clone e usare quello di EsprReale, che
        // implementa una clonazione superficiale;
}
```